第九章 建模与消隐

计算机图形学

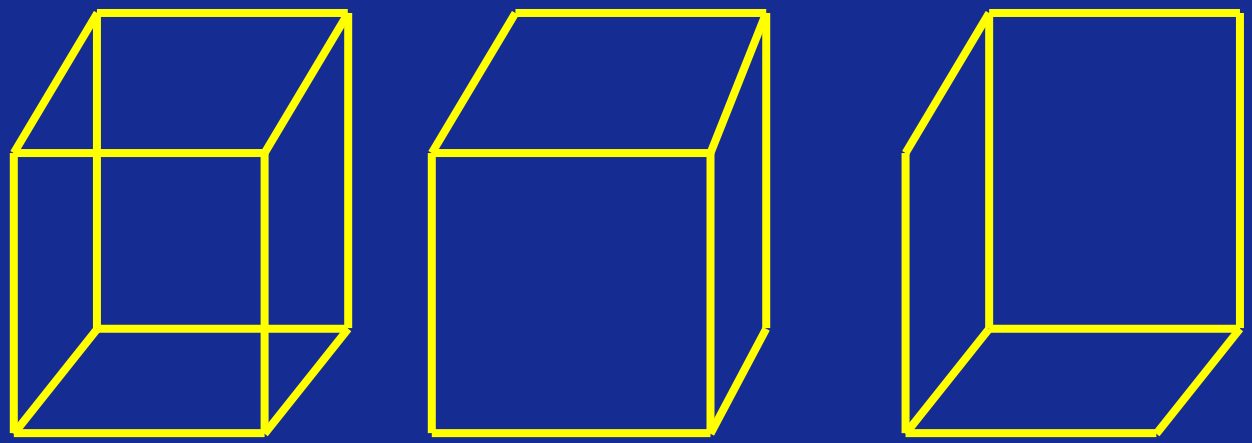
本章主要内容：

- 9.1 三维物体的数据结构
- 9.2 消隐算法分类
- 9.3 隐线算法
- 9.4 隐面算法
- 9.5 本章小结
- 习题9

本章学习目标：

- 了解物体的三种模型表示方法
- 掌握柏拉图多面体的建模方法
- 掌握常见曲面体的建模方法
- 熟练掌握凸多面体背面剔除算法
- 熟练掌握深度缓冲算法
- 熟练掌握深度排序算法

- 在二维显示器上绘制三维图形时，必须把三维信息经过投影变换为二维信息。由于投影变换失去了图形的深度信息，往往导致对图形的理解存在二义性。要生成具有真实感的图形，就要在给定视点和视线方向之后，决定场景中物体哪些线段或表面是可见的，哪些线段或表面是不可见的。这一问题习惯上称为消除隐藏线和消除隐藏面，简称为消隐。



立方体线框模型图

9.1 三维物体的数据结构

- W.K.Giloi在其著作

Interactive Computer Graphics中提出：

Computer Graphics=Data Structure

+Graphics Algorithms+Language

9.1 三维物体的数据结构

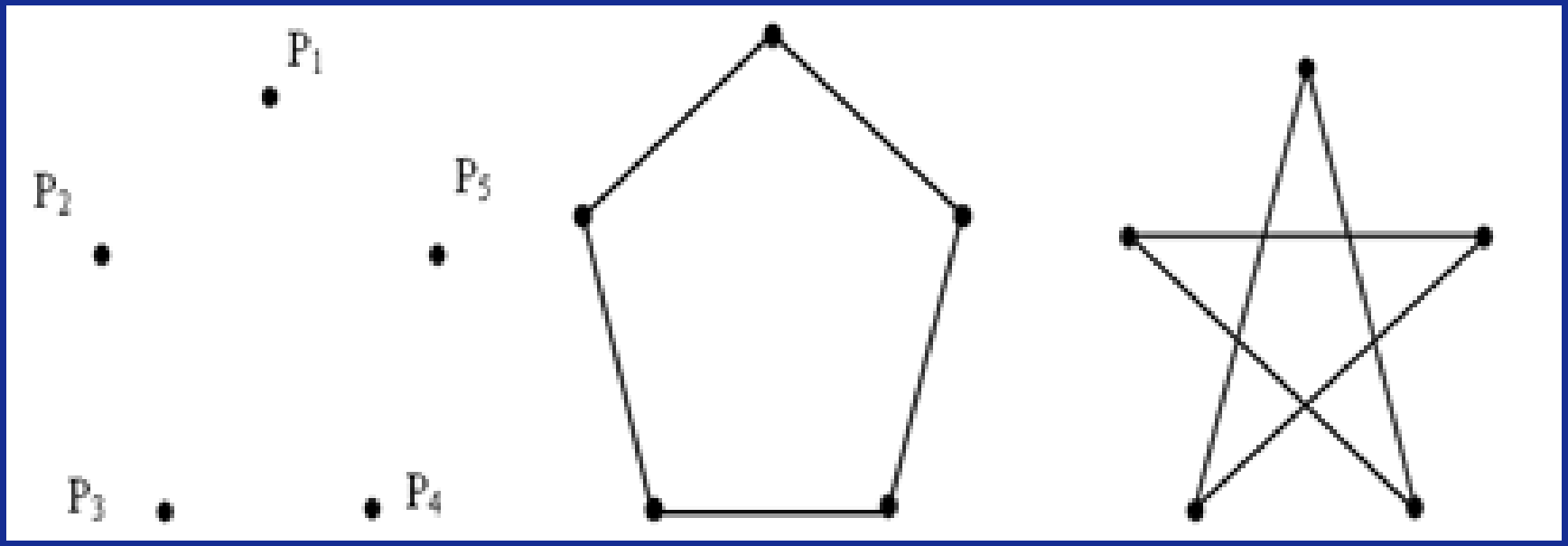
- 场景中经常绘制的三维物体

- 这些物体可以采用线框模型描述，也可以采用表面模型或实体模型描述。无论使用哪种模型描述，都需要为物体建立顶点表、边表和面表构成的数据结构。建立三维用户坐标系为右手系Oxyz，x轴水平向右为正，y轴垂直向上为正，z轴从纸面指向观察者。

9.1.2 物体的几何信息和拓扑信息

- 几何信息：描述几何元素空间位置的信息。
- 拓扑信息：描述几何元素之间相互连接关系的信息。
- 描述一个物体不仅要有几何信息的描述而且还要有拓扑信息的描述。因为只有几何信息的描述，在表示上存在不惟一性。

- 五个顶点，几何信息已经确定，如果拓扑信息不同，则可产生不同图形



9.1.2 三表结构

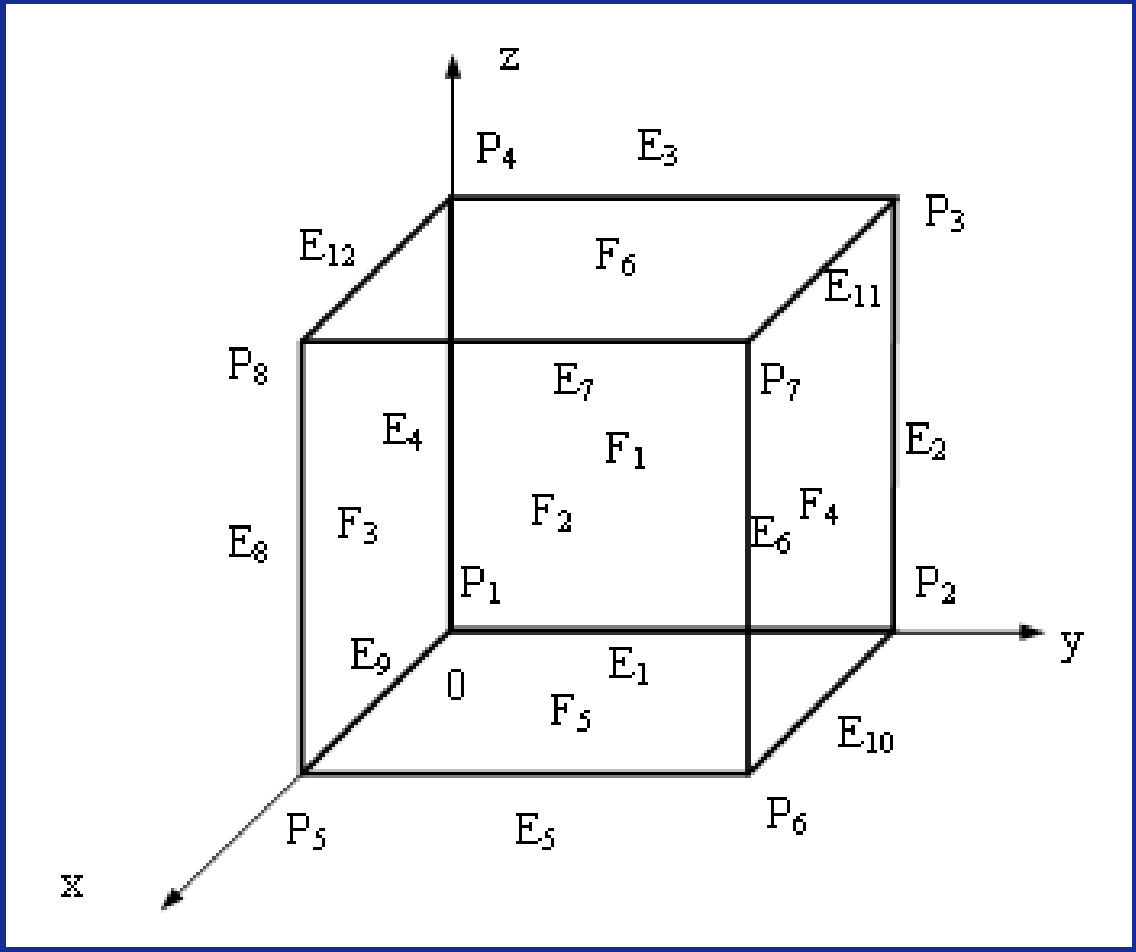
- 在三维坐标系下，描述一个物体不仅需要顶点表，而且还需要边表和面表，才能完全表达清楚

$$V + F - E = 2$$

V表示多面体的顶点数，

F表示多面体的面数，

E表示多面体的边数


z
P4
E3
P3
E12
F6
E11
P8
E7
P7
E4
F1
E2
F3
F2
F4
E8
E6
P1
P2
y
E9
E1
0
F5
E10
P5
E5
P6
x

立体的数据结构

立方体顶点表：立方体顶点的几何信息

顶点	x坐标	y坐标	z坐标
P_1	$x_1=0$	$y_1=0$	$z_1=0$
P_2	$x_2=0$	$y_2=a$	$z_2=0$
P_3	$x_3=0$	$y_3=a$	$z_3=a$
P_4	$x_4=0$	$y_4=0$	$z_4=a$
P_5	$x_5=a$	$y_5=0$	$z_5=0$
P_6	$x_6=a$	$y_6=a$	$z_6=0$
P_7	$x_7=a$	$y_7=a$	$z_7=a$
P_8	$x_8=a$	$y_8=0$	$z_8=a$

立方体边表：立方体边的拓扑信息

边	起点	终点
E_1	P_1	P_2
E_2	P_2	P_3
E_3	P_3	P_4
E_4	P_4	P_1
E_5	P_5	P_6
E_6	P_6	P_7
E_7	P_7	P_8
E_8	P_8	P_5
E_9	P_1	P_5
E_{10}	P_2	P_6
E_{11}	P_3	P_7

立方体面表：立方体面的拓扑信息

面	边1	边2	边3	边4
F_1	E_1	E_4	E_3	E_2
F_2	E_5	E_6	E_7	E_8
F_3	E_9	E_8	E_{12}	E_4
F_4	E_{10}	E_2	E_{11}	E_6
F_5	E_1	E_{10}	E_5	E_9
F_6	E_3	E_{12}	E_7	E_{11}

9.1.3 物体三维表示模型

- 线框模型
- 表面模型
- 实体模型

线框模型

- 线框模型只使用顶点表和边表两个数据结构描述面点之间的拓扑关系

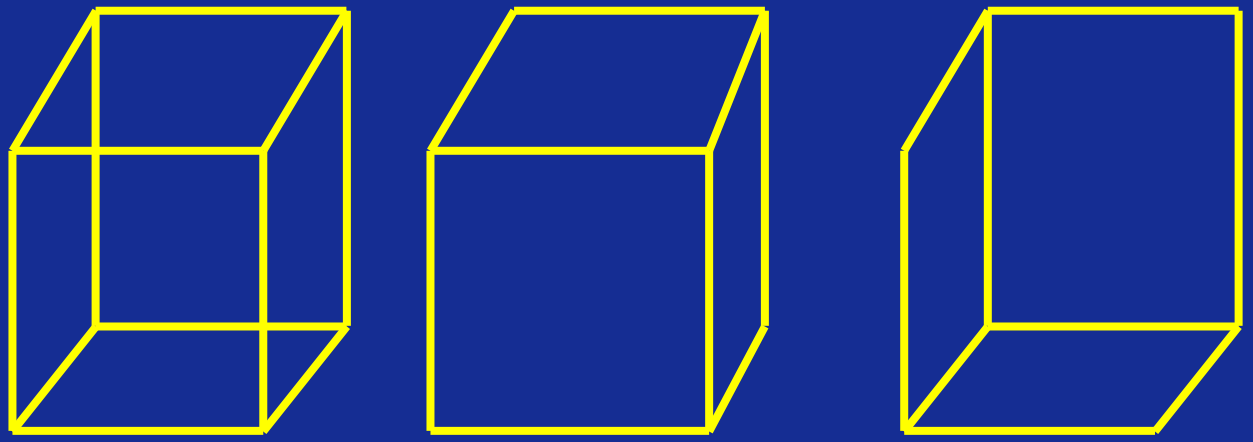
- 线框模型只是用几何体的边线来表示立体的外形，就如同用边线搭出的框架一样，线框模型中没有表面、体积等信息。

线框模型

- 线框模型的优点：可以产生任意方向视图，视图间能保持正确的投影关系
- 线框模型缺点：

  因为所有棱边全部绘制出来，理解方面容易产生二义性

  线框模型不能进行两个面的求交运算



线框模型表示的二义性

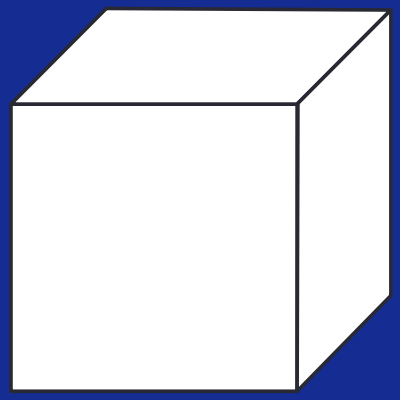
表面模型

- 表面模型是利用立体的外表面来构造模型
- 就如同在线框模型上蒙上了一层外皮，使立体具有了一定的轮廓，可以进行消隐处理
- 与线框模型相比，表面模型增加了一个面表（无说明），用以记录边面之间的拓扑关系。

表面模型

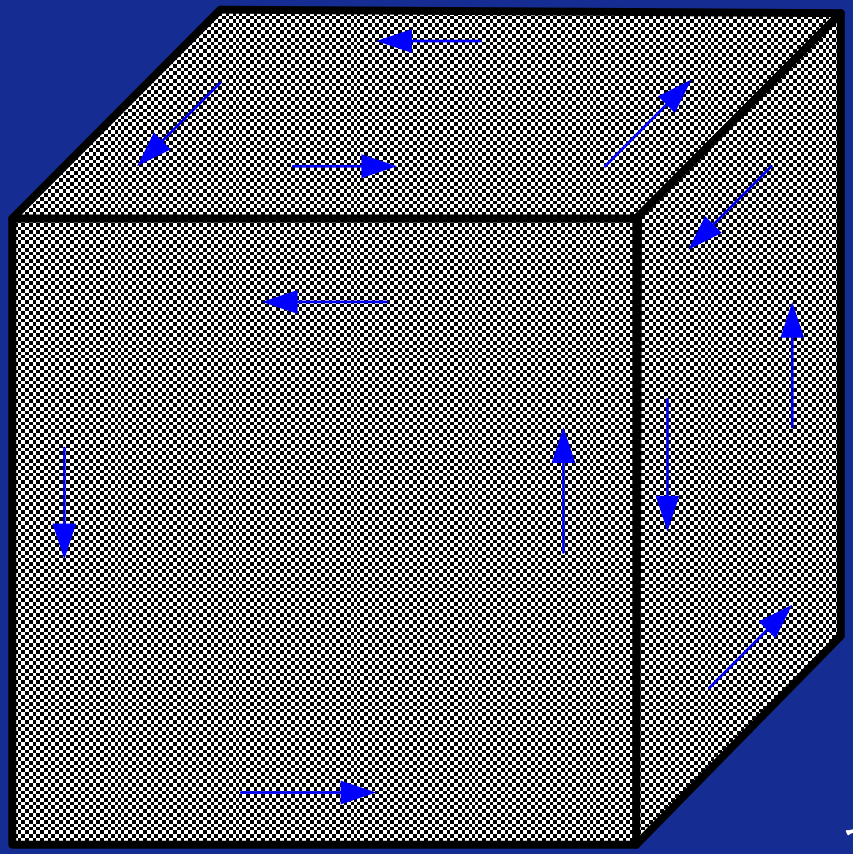
- 表面模型的优点：可以进行面着色，隐藏面消隐，以及表面积计算，有限元网格划分等。
- 表面模型缺点：
  - 表面模型仍缺乏体积的概念，

    是一个立体的空壳
  - 无法进行立体之间的相交运算



实体模型

- 拓扑合法的立体在相邻两个面的公共边界上，棱边的方向正好相反



立方体实体模型表示

实体模型

- 实体模型，如同在立体表面模型的内部进行了填充，使之具有了如体积和重量等特性，更能反映立体的真实性，这时的立体才具有"体"的概念。
- 实体模型与表面模型的不同之处在于确定了表面的哪一侧存在实体
- 在表面模型的基础上采用有向棱边隐含地表示表面的外法矢方向，使用右手法则取向：四个手指沿闭合的棱边方向，大拇指方向与表面外法矢量方向一致

实体模型

- 实体模型只需将面表改成环表形式，就可确切地分清体内体外
- 实体模型和线框模型、表面模型的根本区别在于其数据结构不仅记录了全部几何信息，而且记录了全部点、线、面、体的拓扑信息
- 实体模型常采用集合论中的并、交、差等运算来构造复杂形体

消隐算法分类

- 无论多么复杂的空间物体，沿视线方向上都只能看到一部分表面，其余的表面背向观察者不可见
- 计算机图形学的一个重要任务就是对空间物体的表面进行可见性检测，绘制出可见边线和表面
- 经过消隐后得到的图形称为消隐图

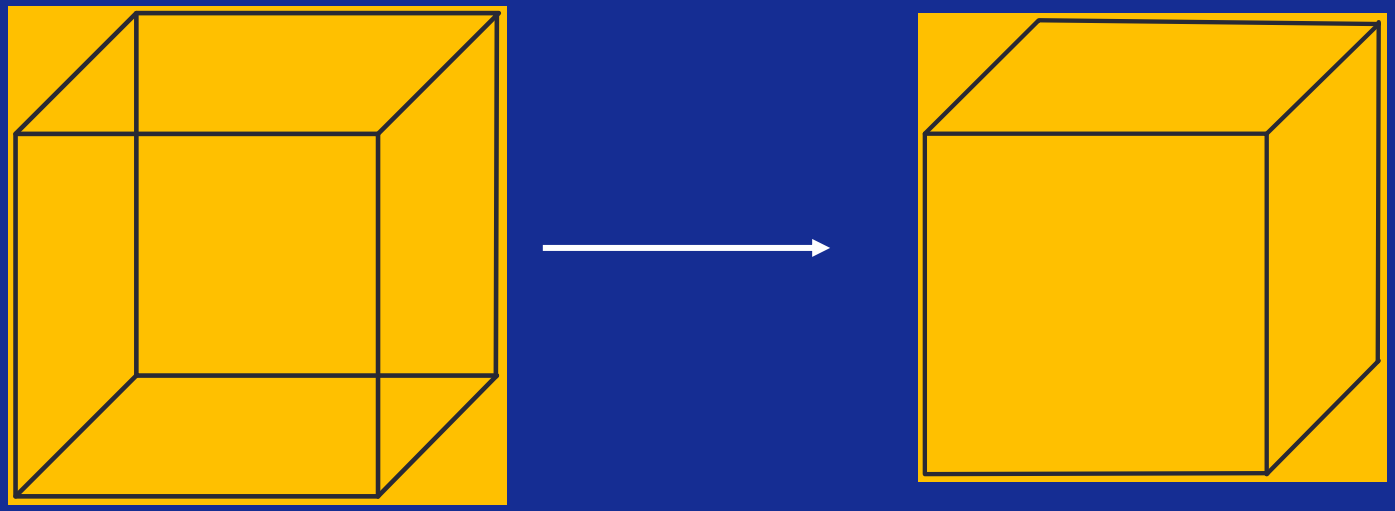
- 根据消隐方法的不同，消隐算法可分为两类：
  - 隐线算法

    用于消除物体上不可见的边界线

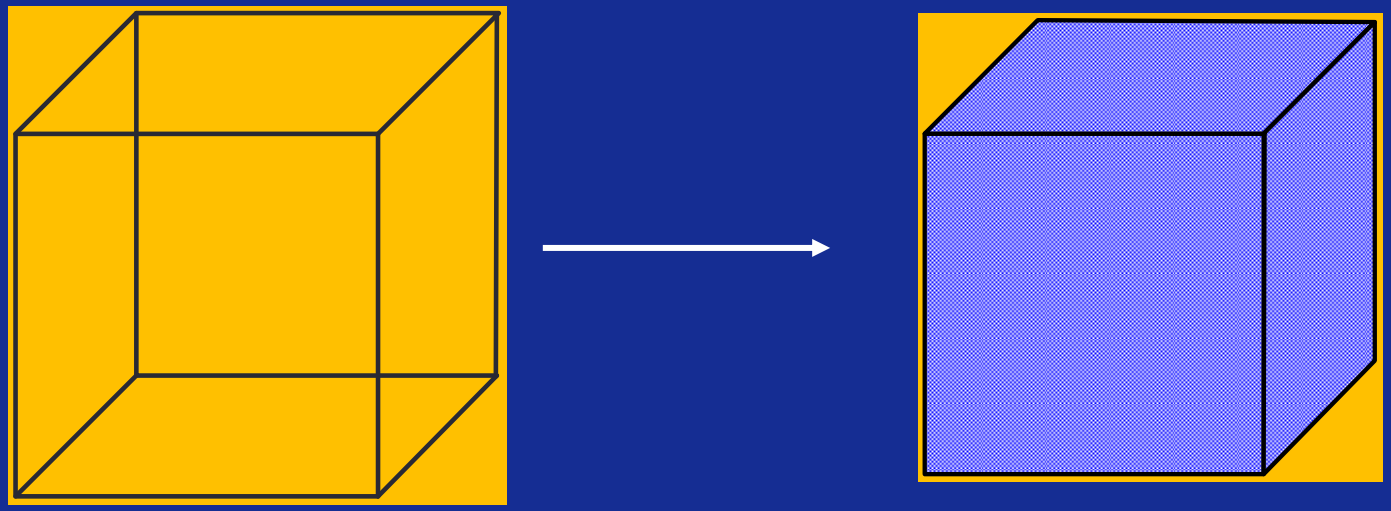
    主要针对线框模型，要求画出物体的各可见棱边
  - 隐面算法

    用于消除物体上不可见的表面

    主要针对表面模型，不仅要求画出物体的各个可见棱边，而且还要求填充各个表面



隐线算法



隐面算法

- 计算机图形学的创始人Southerland根据消隐空间的不同，将消隐算法分为三类：
  - 物空间消隐算法：在描述物体的坐标系空间中进行消隐。
  - 像空间消隐算法：在物体投影后的二维图像空间中进行消隐。
  - 物像空间消隐算法：在描述物体的坐标系空间和图像空间同时进行消隐。

9.3 隐线算法

- 线框模型消隐一般在物空间中进行
- 物空间消隐算法是根据边界线的可见性检测条件，判断哪些边界线是可见的，哪些边界线是不可见的，在屏幕上只绘制可见边界线。

9.3.1 凸多面体隐线算法

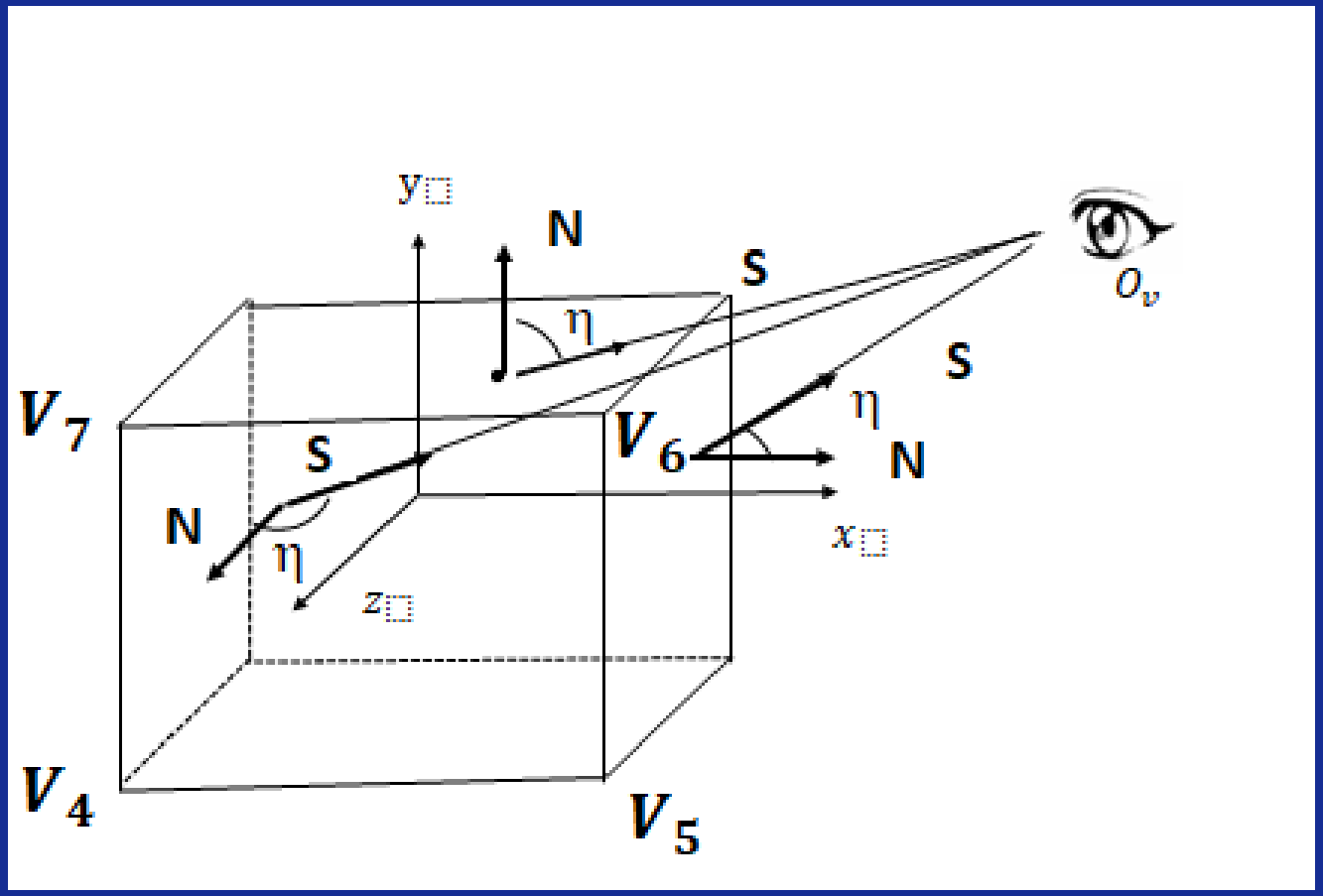
- 在消隐问题中，凸多面体消隐是最简单和最基本的情形。凸多面体具备这样的性质：连接立体上不同表面的任意两点的直线完全位于该凸多面体之内。凸多面体由凸多边形构成，其表面要么完全可见，要么完全不可见。凸多面体消隐算法的关键是给出测试其表面边界线可见性的判别式。
- 事实上，对于凸多面体的任一个面，可以根据其外法矢量和视矢量的夹角 η 来进行可见性检测。如果两个矢量的夹角 $0° \le \eta < 90°$ 时，表示该表面可见；如果 $90° < \eta \le 180°$ 时，表示该表面不可见。

9.3.1 凸多面体隐线算法

- 事实上凸多面体隐线算法也是一种隐面算法，是通过判断表面的可见性后才绘制面的边界，只不过该方法可以用于绘制线框模型，才称为隐线算法。

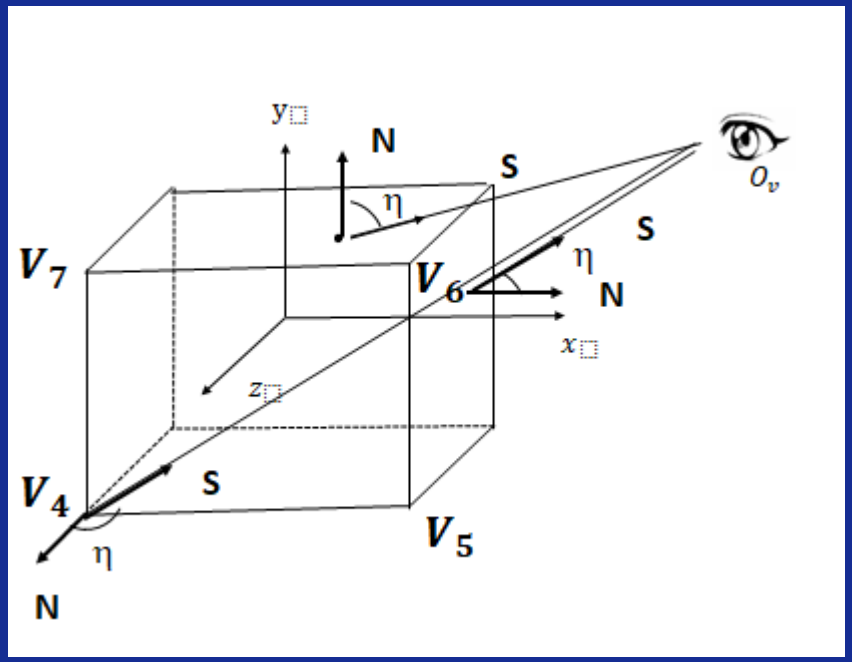
隐线算法

- 外法矢量N， 视矢量为S，外法矢量和视矢量的夹角为η



隐线算法

- 在取"前面"$V_4V_5\ V_6\ V_7$ 取V_4点为参考点



该点的外法矢量为两个边矢量的矢量积 $N=\overrightarrow{V_4V_5}\times\overrightarrow{V_4V_6}$

隐线算法

- 已知点$\boldsymbol{V_4}(x_4, y_4, z_4)$、点$\boldsymbol{V_5}(x_5, y_5, z_5)$和点$\boldsymbol{V_6}(x_6, y_6, z_6)$坐标

边矢量$\overrightarrow{V_4V_5}$的三个分量为：

$$F_x = x_5 - x_4, F_y = y_5 - y_4, F_z = z_5 - z_4$$

边矢量$\overrightarrow{V_4V_6}$的三个分量为：

$$H_x = x_6 - x_4, H_y = y_6 - y_4, H_z = z_6 - z_4$$

隐线算法

- 外法矢量为两个边矢量的矢量积 $N = \overrightarrow{V_4V_5} \times \overrightarrow{V_4V_6}$

外法矢量的三个分量为：

$$N_x = F_y \cdot H_z - F_z \cdot H_y \qquad N_y = F_z \cdot H_x - F_x \cdot H_z \qquad N_z = F_x \cdot H_y - F_y \cdot H_x$$

因此，“前面”外法矢量表示为：

$$N = N_x i + N_y j + N_z k$$

式中，i，j，k分别为三维坐标系的3个单位矢量。

隐线算法

- 给定视点位置球面坐标表示为Ov($R\sin\varphi\sin\theta$, $R\cos\varphi$, $R\sin\varphi\cos\theta$)
- 视矢量从多边形的参考点V_4指向视点，视矢量的计算公式为

$$S=\left(R\sin\varphi\sin\theta-x_4, R\cos\varphi-y_4, R\sin\varphi\cos\theta-z_4\right)$$

- 视矢量表示为

$$S=S_x i+S_y j+S_z k$$

式中，i，j，k分别为三维坐标系的3个单位矢量。

隐线算法

- 表面外法矢量和视矢量的数量积为：

$$N \cdot S = N_x \cdot S_x + N_y \cdot S_y + N_z \cdot S_z$$

$$\cos\eta = \frac{N \cdot S}{|N| \cdot |S|} = \frac{N_x \cdot S_x + N_y \cdot S_y + N_z \cdot S_z}{|N| \cdot |S|}$$

- 将外法矢量和视矢量分别规范化为单位矢量n和s，则有

$$n \cdot s = \cos\eta = n_x \cdot s_x + n_y \cdot s_y + n_z \cdot s_z$$

- $\cos\eta$ 的正负取决于表面的单位法外矢量与单位视矢量的数量积

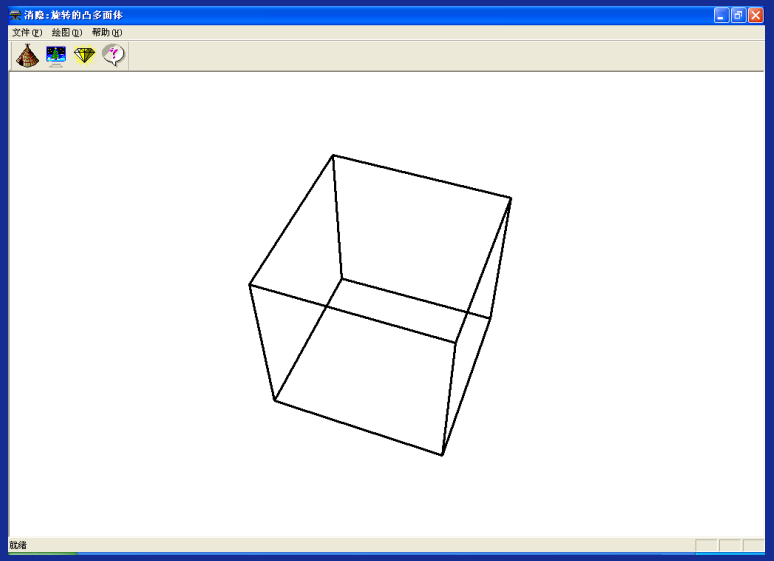
即 $n_x \cdot s_x + n_y \cdot s_y + n_z \cdot s_z$

隐线算法

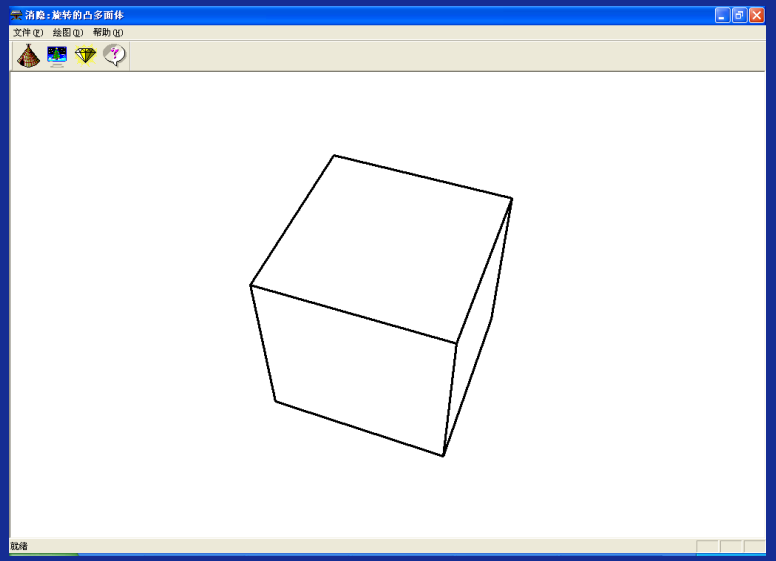
- 在凸多面体表面可见性检测条件为：
- 当0°≤ η 90°时， $n_x \cdot s_x + n_y \cdot s_y + n_z \cdot s_z \succ 0$ 方体表面可见，绘该表面多边形；
- 当 η =90°时， $n_x \cdot s_x + n_y \cdot s_y + n_z \cdot s_z = 0$ 表面外法矢量与视矢量垂直，立方体表面多边形退化为一条直线，绘出该直线；
- 当90° < η 80°时， $n_x \cdot s_x + n_y \cdot s_y + n_z \cdot s_z \prec 0$ 方体表面不可见，不绘制该多边形。

背面剔除隐线算法（Back culling）

- 对于立方体而言，使用了 $n \cdot s \geq 0$ 作为绘制可见表面边界的基本条件，即剔除了背向视点的三个不可见表面，只绘制朝向视点的3个可见表面。因此，本算法又被称之为背面剔除算法。



消隐前的立方体



消隐后的立方体

9.4 隐面算法

- 隐面算法是指从视点的角度观察物体的表面，离视点近的表面遮挡了离视点远的表面，屏幕上绘制的结果为所有可见表面最终投影的集合。
- 一种方法是表面的投影顺序有关，在屏幕上先投影离视点远的表面，再投影离视点近的表面，后绘制的表面遮挡了先绘制的表面，称为深度排序算法；另一种方法与表面投影顺序无关，但使用了缓冲器记录了物体表面在屏幕上投影所覆盖范围内的全部像素的深度值，因此访问屏幕范围内物体表面所覆盖的每一像素，用深度小（深度用z值表示，z值小表示离视点近）的像素点颜色代替深度大的像素点颜色可以实现消隐，称为深度缓冲器算法。

9.4.1 Z-Buffer算法

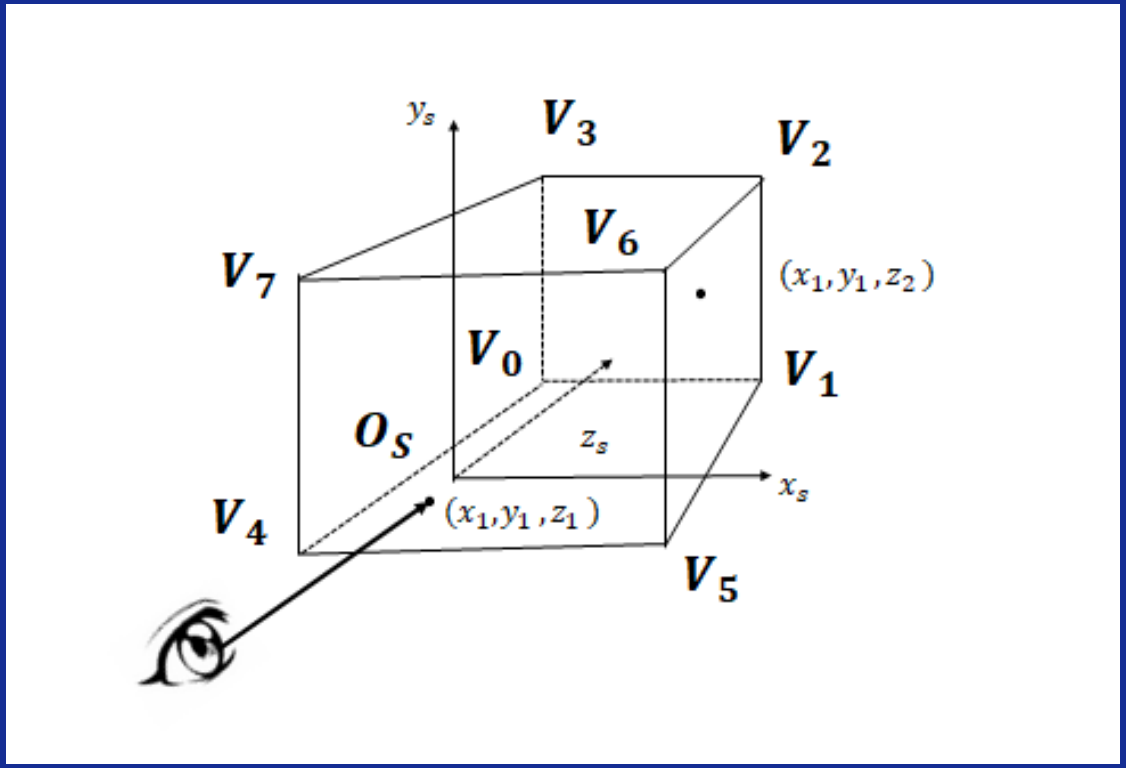
- Catmull在1974年提出了Z-Buffer算法
- 在像空间中根据深度值确定最终绘制的立体各个表面的像素点颜色
- Z-Buffer算法也称深度缓冲算法，通常x，y代表屏幕坐标而z则用来表示立体上各个面的深度，故名Z-Buffer算法

9.4.1 Z-Buffer算法

- Catmull在1974年提出了Z-Buffer算法
- 在像空间中根据深度值确定最终绘制的立体各个表面的像素点颜色
- Z-Buffer算法也称深度缓冲算法，通常x，y代表屏幕坐标而z则用来表示立体上各个面的深度，故名Z-Buffer算法

Z-Buffer算法

- 建立三维屏幕坐标系如图所示
- 观察者的视点位于z_s轴的负方向，视线方向沿着z_s轴正方向，指向$x_sO_sy_s$坐标面

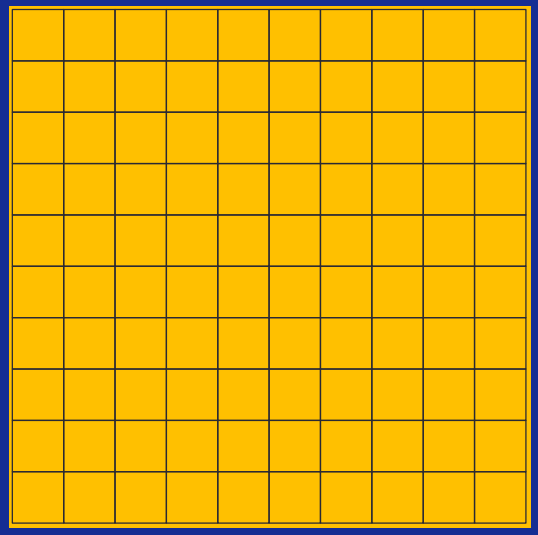


Z-Buffer算法

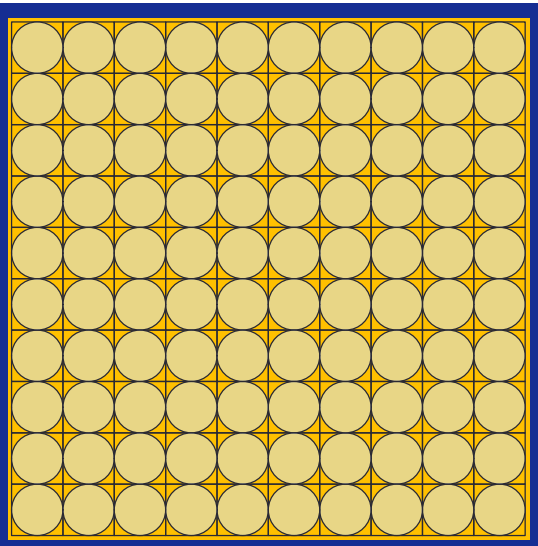
- 对于图示立方体，z轴与立方体的前面交于（x_1,y_1,z_1），与立方体的后面交于（x_1,y_1,z_2）
- 前面和后面在屏幕上的投影坐标（x_1,y_1）相同，但z1<z2，（x_1,y_1,z_1）离观察者近，（x_1,y_1,z_2）离观察者远
- 对于投影像素点（x_1,y_1），前面像素点（x_1,y_1,z_1）的颜色将覆盖后面像素点（x_1,y_1,z_2）的颜色，显示结果为前面像素点（x_1,y_1,z_1）的RGB颜色

Z-Buffer算法

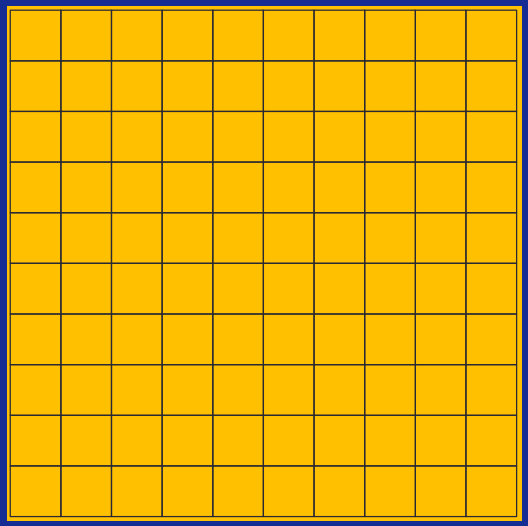
- 这样Z-Buffer算法需要建立两个缓冲器：
  1. 深度缓冲器，用于记录立体上每一个像素的深度值，初始化为立体的最大深度值（z坐标）
  2. 帧缓冲器，用于记录立体上每个像素点的颜色值，初始化为屏幕的背景色



帧缓冲器



屏幕像素点



深度缓冲器（Z缓冲器）

Z-Buffer算法

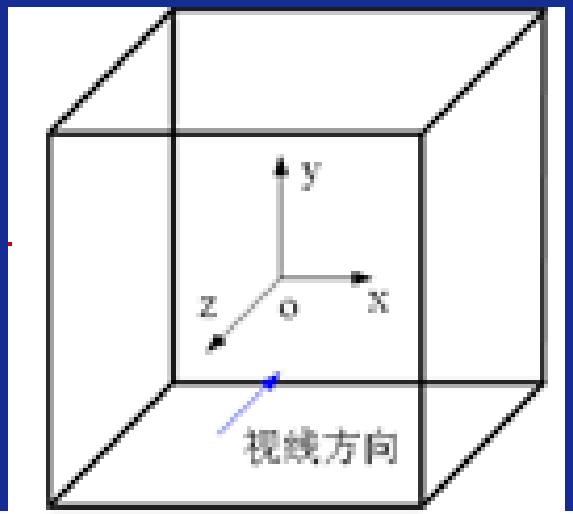
- Z-Buffer算法对立体的各个表面全部进行采样，将采样点变换到屏幕坐标系的像空间，并记录其深度z坐标值。
- 根据采样点在屏幕上的投影位置，将其深度与已存储在帧缓存器中相应像素点处的深度值进行比较
  - 如果新采样点的深度值小于原可见点的深度值，表明新采样点更靠近观察者，遮住了原采样点，则用新采样点处的颜色值更新帧缓存器中相应采样点处的颜色值，同时用新采样点的深度值更新深度缓存器中的采样点的深度值；
  - 否则，不作更改。

Z-Buffer算法

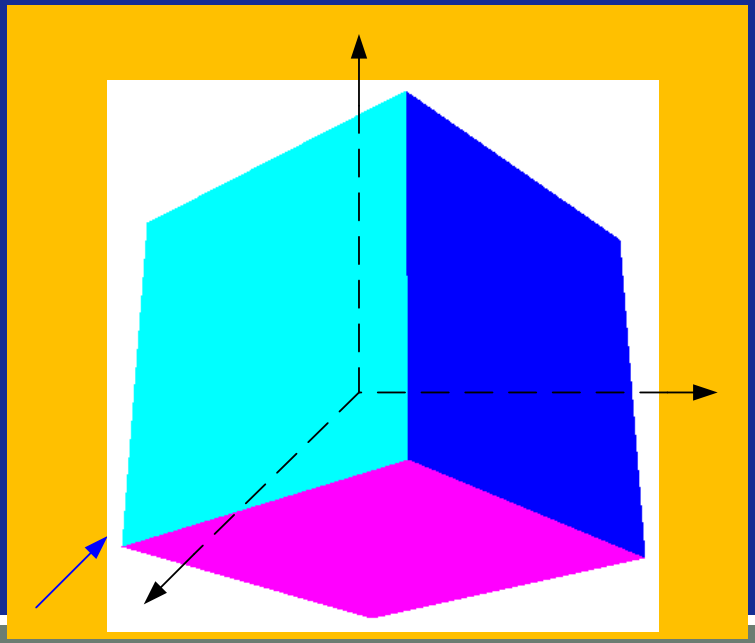
- 对一给定视线上的(x_s, y_s)，查找离视点最近的 $z_s(x_s, y_s)$值。

- 帧缓冲器初始值置为背景色；
- 确定深度缓冲器的宽度、高度和初始深度。一般初始深度为最大深度值
- 对于多边形表面的每一像素(x_s, y_s)，计算其深度值$z_s(x_s, y_s)$
- 将$z_s(x_s, y_s)$与存储在深度缓冲器中的(x_s, y_s)处的深度值$zBuffer(x_s, y_s)$进行比较。
- 如果$z_s(x_s, y_s) \leq zBuffer(x_s, y_s)$，则将此像素的颜色写入帧缓冲器，且用$z_s(x_s, y_s)$重置$zBuffer(x_s, y_s)$

- Z-Buffer算法实现的难点是如何计算采样点的深度值z（x，y）
- 对图所示的静止立方体，立方体前面和后面都平行于xoy坐标面，上面、下面、左面和右面四个表面皆与屏幕坐标系的xoy面垂直，立方体的投影为正方形，这时，前面和后面的深度值很容易计算，立方体的消隐结果为正方形。



- 旋转立方体的动态Z-Buffer消隐算法
- 当立方体处于图所示的位置时，六个面都不与屏幕平行，需要根据每个面的平面方程计算其上各个像素点深度值，这里需要用到第四章已经讲解的有效边表算法来访问面上的每一个像素点进行消隐。

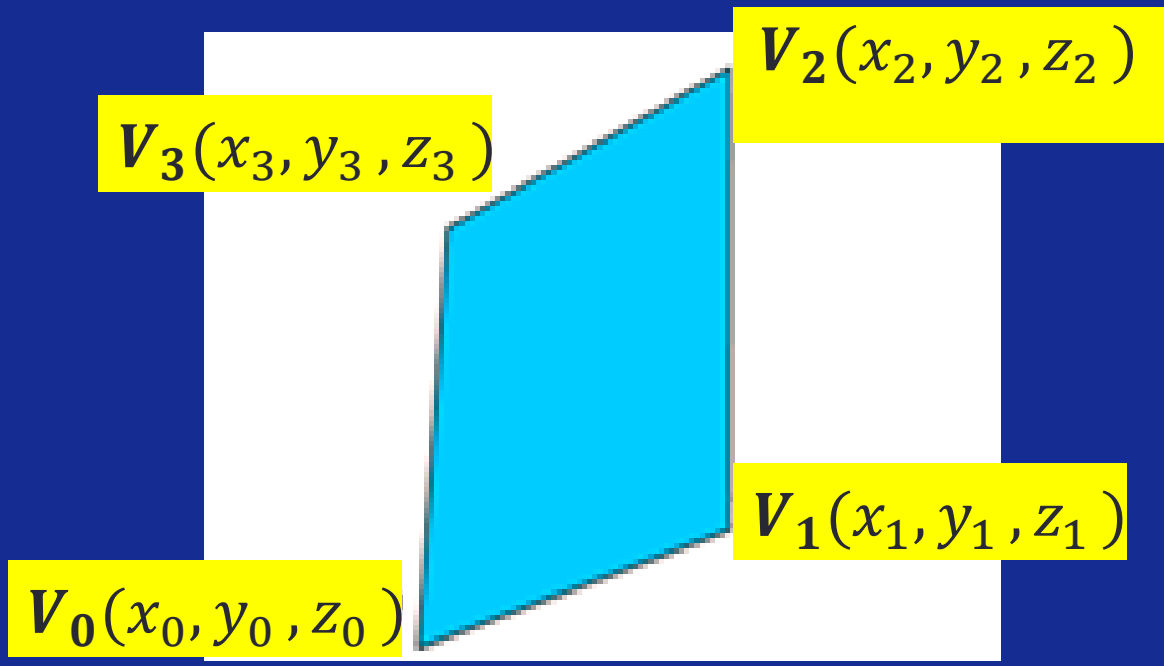


- 对于图所示的立方体表面，其平面一般方程为：

$$Ax + By + Cz + D = 0$$

- 式中系数A，B，C是该平面的一个法矢量的坐标，即

$$N = \{A, B, C\}$$



- 根据表面顶点的坐标值可以计算两个边矢量

矢量 $\overrightarrow{V_0V_1}=\{x_1-x_0,y_1-y_0,z_1-z_0\}$

矢量 $\overrightarrow{V_0V_2}=\{x_2-x_0,y_2-y_0,z_2-z_0\}$

- 根据两个边矢量的叉积，可求系数A，B，C

$$A=(y_1-y_0)\cdot(z_2-z_0)-(z_1-z_0)\cdot(y_2-y_0)$$

$$B=(z_1-z_0)\cdot(x_2-x_0)-(x_1-x_0)\cdot(z_2-z_0)$$

$$C=(x_1-x_0)\cdot(y_2-y_0)-(y_1-y_0)\cdot(x_2-x_0)$$

- 将A、B、C和点（x_0,y_0,z_0）代入前面方程

$$D=-Ax_0-By_0-Cz_0$$

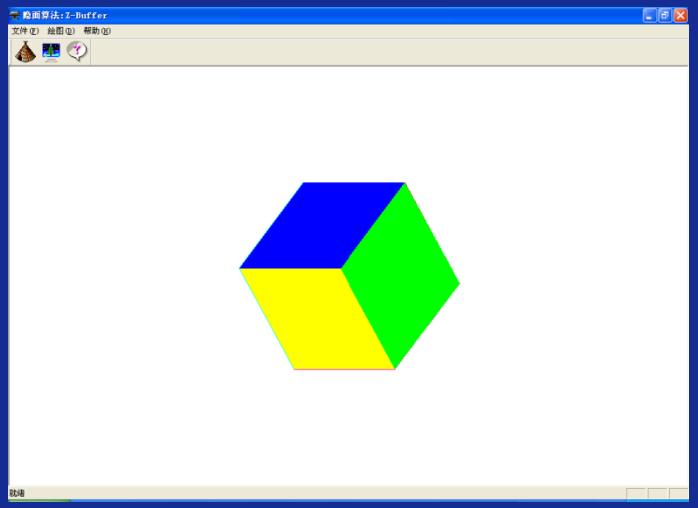
- 从方程可以得到当前像素点（x_s，y_s）处的深度值：

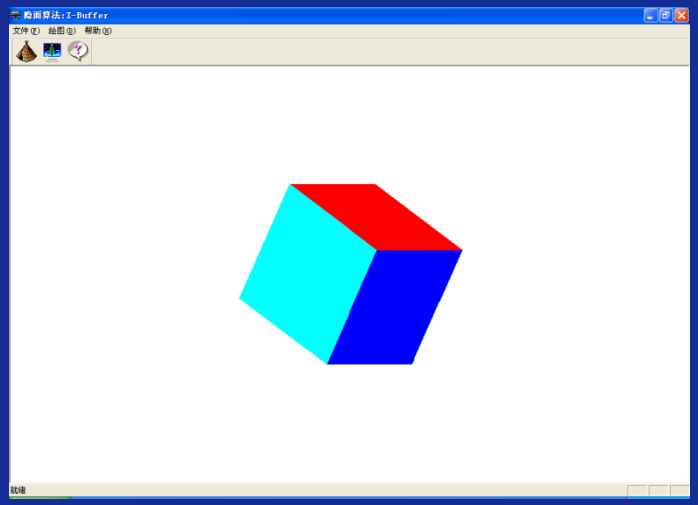
$$z_s(x_s, y_s) = \frac{-Ax_s - By_s - D}{C}$$

- 这里，如果C = 0，说明表面的法矢量与z轴垂直，在xoy面内的投影为一条直线，在算法中可以不予以考虑。
- 如果已知扫描线y_i与多边形相交的像素点 (x_i, y_i)处的深度值$z_s(x_i$ ，y_i)，其相邻点(x_{i+1} ，y_i)处的深度值为z(x_{i+1}, y_i)

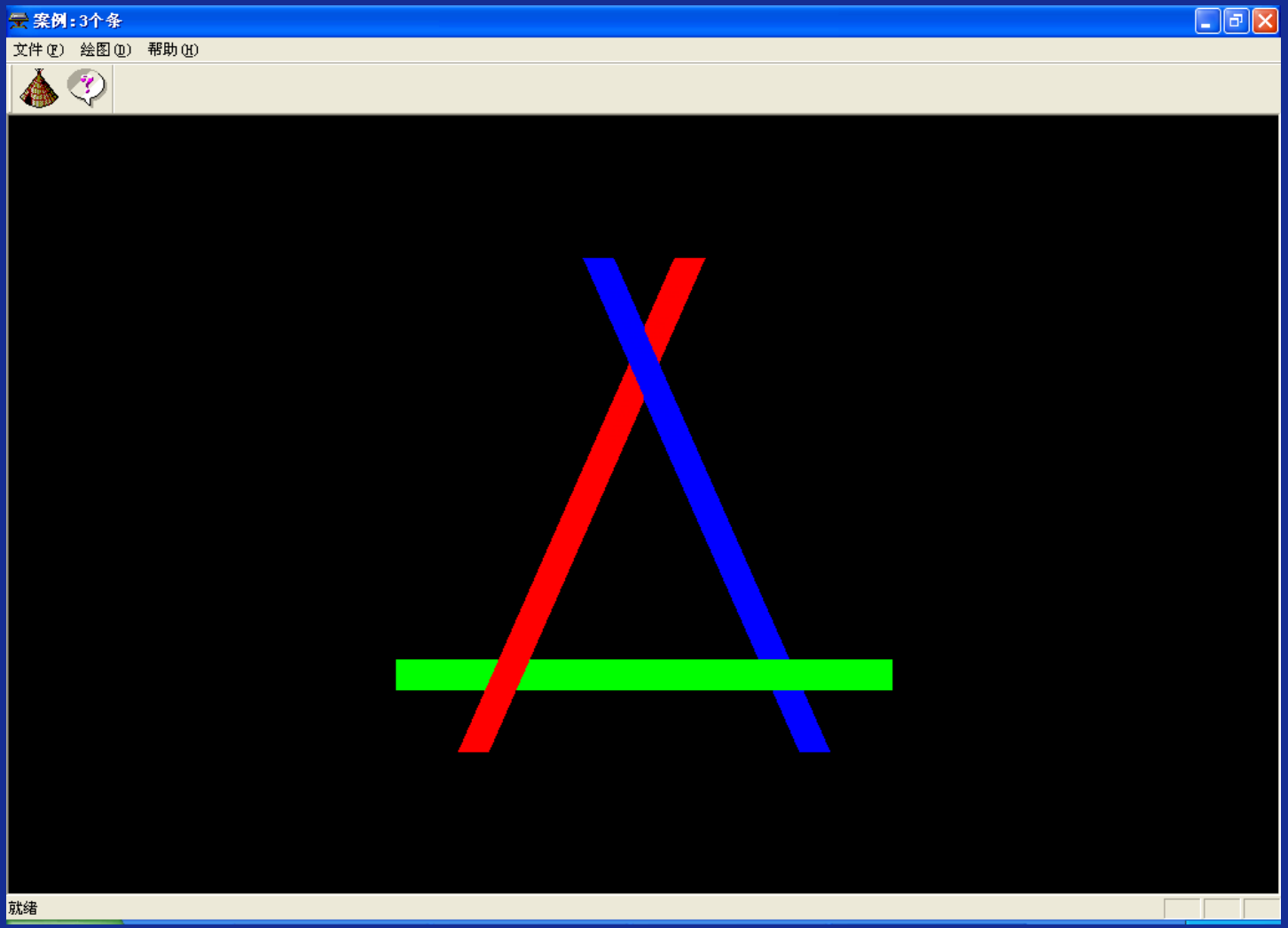
$$z(x_i + 1, y_i) = \frac{-A(x_i + 1) - By_i - D}{C} = z(x_i, y_i) - \frac{A}{C}$$

- 计算出多边形每个面上各个像素点的深度值后，就可以使用Z-Buffer算法进行消隐，旋转立方体消隐效果如图所示。





- Z-Buffer算法的优点在于简单，有利于硬件实现。由于物体表面可以按照任意次序写入帧缓冲器和深度缓冲器，故无需按深度优先级排序，节省了排序时间。
- 算法的缺点是需要占用大量的存储单元。
- 实际中一般很少将深度缓冲器的宽度和高度取为屏幕客户区的大小，而是先检测物体表面全部投影所覆盖的最大范围，以确定深度缓冲器数组的大小，可以有效地减少存储容量。

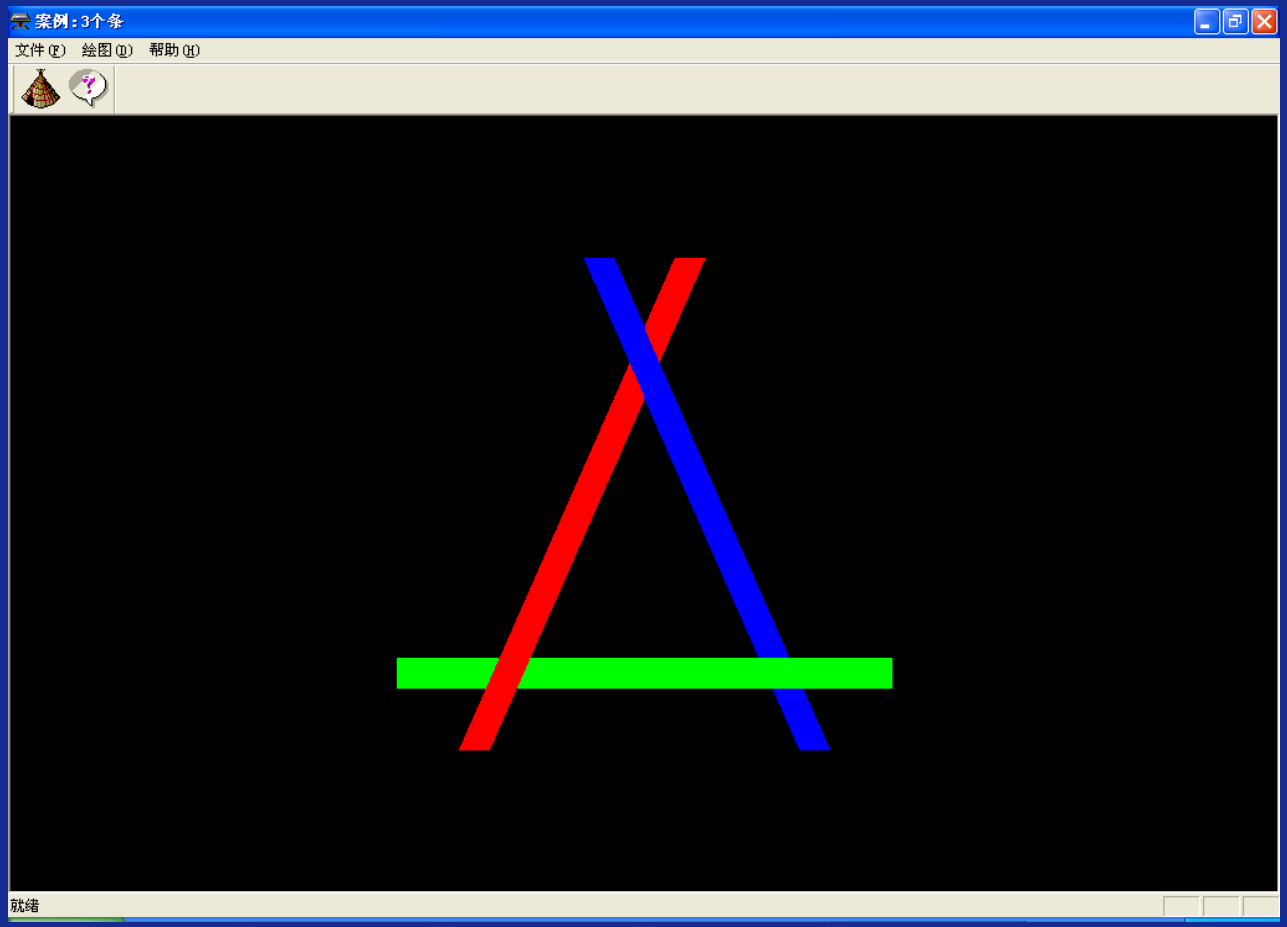

案例：3个条
文件(F)　绘图(D)　帮助(H)
就绪

9.4.2 深度排序算法

- 同时运用了物体空间和图像空间的消隐算法。在物体空间将物体表面按深度优先级排序，然后在图像空间从深度最大的表面开始，依次绘制各个表面。这种消隐算法又称之为画家算法。
- 画家在创作一副油画时，总是先画远景，再画中景，最后才画近景。这样不同的颜料将依次堆积覆盖，形成层次分明的艺术作品。

画家算法

- 先把屏幕置成背景色
- 再把物体的各个面按其离视点的远近进行排序，排序结果存在一张深度优先级表中：离视点远者深度大（z值）位于表头，离视点近者深度小（z值）位于表尾。
- 然后按照从表头到表尾（从远到近）的顺序，逐个取出逐个绘制各个面，后绘制的表面颜色取代先绘制的表面颜色，相当于消除了隐藏面。


案例：3个条
文件(F)　绘图(D)　帮助(H)
就绪

画家算法

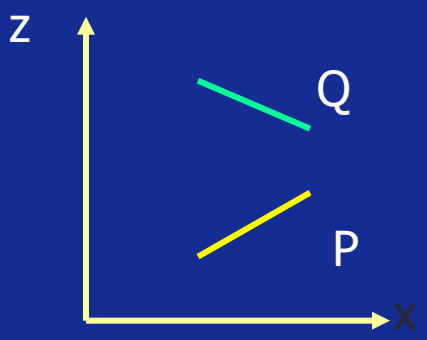
- 关键是如何对场景中的物体按深度排序

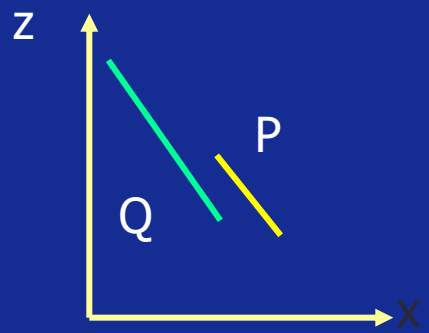
根据每个多边形的Zmin对它们预排序；

深度重叠测试:

Zmin(P)<Zmin(Q)，若Zmax(P)<Zmin(Q)，则P肯定不能遮挡Q；

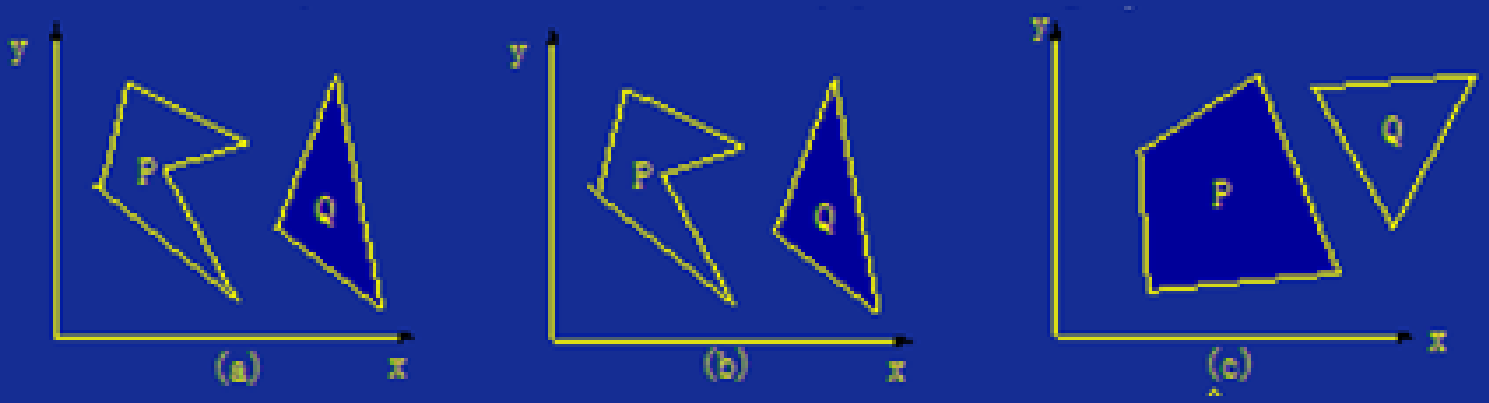
若Zmax(P)>Zmin(Q)，则进一步检查；





画家算法

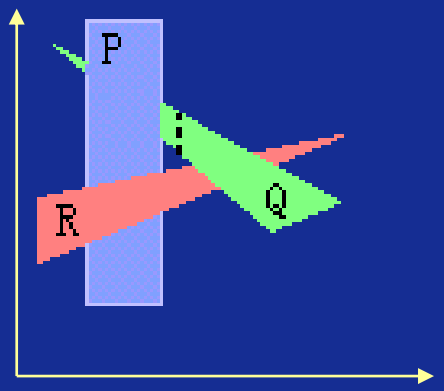
- 投影重叠判断
  - P和Q在oxy平面上投影在x方向上不相交
  - P和Q在oxy平面上投影在y方向上不相交
  - P和Q在oxy平面上的投影不相交
  - P在Q之后。P的各顶点均在Q的远离视点的一侧
  - Q在P之前。Q的各顶点均在P的靠近视点的一侧
  - **只要有一项成立，P就不遮挡Q**

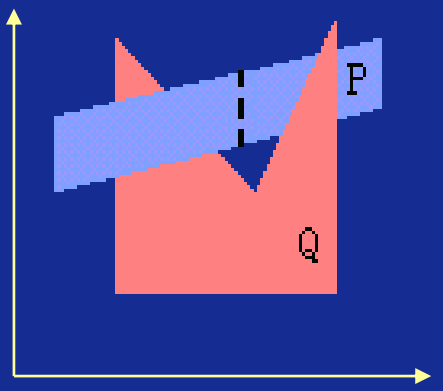


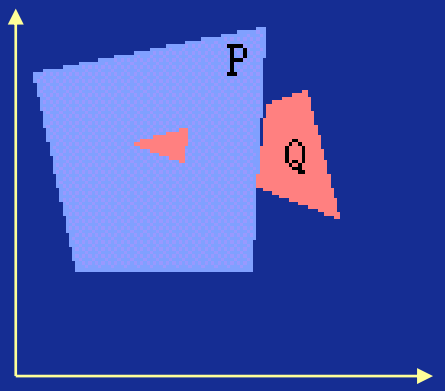
画家算法

精确重叠判断

- 颜色重叠，须作进一步判断：对P、Q在xoy平面上的投影做求交运算。计算时不必具体求出重叠部分。在交点处进行深度比较，只要能判断出前后顺序即可。若遇到多边形相交或循环重叠的情况，还必须在相交处分割多边形，然后进行判断。







9.5 本章小结

- 主要讲述多面体与曲面的几何模型的方法。根据给出的顶点表和表面表可以绘制物体的三维模型。
- 三维图形的消隐算法分为隐线算法和隐面算法。
- 其中隐线算法主要针对线框模型（多面体）或网格模型（曲面体）进行，只根据表面法矢量和视矢量的点积就能判别表面是否可见。
- 隐面算法重点讲解了Z-Buffer算法，计算机图形学中最主要的消隐算法，也是OpenGL中唯一使用的隐面算法。在面消隐之前，一般先对物体的多边形表面进行背面剔除预处理，然后才对可见表面使用Z-Buffer算法，从像素级角度对物体进行消隐。

习题9

- 课后自行完成
- 7.ZBuffer算法的代表案例绘制是三个相互叠压的红绿蓝条